Experiences Using Formal Methods for Requirements Modeling

Steve Easterbrook
NASA IV&V Facility,
100 University Drive, Fairmont, West Virginia 26505,
steve@atlantis.ivv.nasa.gov

Robyn Lutz, Rick Covington, John Kelly
NASA Jet Propulsion Lab, Pasadena, California

Yoko Ampo
NEC Corp, Tokyo, Japan

and

David Hamilton
Hewlett Packard Corp, San Diego, California

Abstract

This paper describes three cases studies in the lightweight application of formal methods to requirements modelling for spacecraft fault protection systems. The case studies differ from previously reported applications of formal methods in that formal methods were applied very early in the requirements engineering process, to validate the evolving requirements. The results were fed back into the projects, to improve the informal specifications. For each case study, we describe what methods were applied, how they were applied, how much effort was involved, and what the findings were. In all three cases, the formal modeling provided a cost effective enhancement of the existing verification and validation processes. We conclude that the benefits gained from early modeling of unstable requirements more than outweigh the effort needed to maintain multiple representations.

Experiences Using Formal Methods for Requirements Modeling

Steve Easterbrook (NASA IV&V Facility, 100 University Drive, Fairmont West Virginia),
Robyn Lutz, Rick Covington, John Kelly (NASA Jet Propulsion Lab, Pasadena, California),
Yoko Ampo (NEC Corp, Tokyo, Japan)
and **David Hamilton** (Hewlett Packard Corp, San Diego, California)

I. Introduction

Recent studies have indicated that formal methods can offer significant benefits in improving the safety and reliability of large software systems [1]. However, despite the occasional success story, the uptake of formal methods has been slow. Part of the problem seems to be a chasm between the work on formal methods described in the literature and the needs of industry [2]. In this paper we present three case studies of successful application of formal methods for requirements modeling. The studies demonstrate that a pragmatic, lightweight application of formal methods can offer a cost-effective way of improving the quality of software specifications.

There is an emerging consensus that "formal methods seem to find their most effective application early in the lifecycle, where conventional methods are apparently weakest."[3]. Studies of change requests to the Space Shuttle flight software have demonstrated that formal methods are particularly good at improving the clarity and precision of requirements specifications, and in finding important and subtle errors [4 6]. This benefit coincides with a serious, unmet need in developing embedded, mission-critical software: the need for early feedback on the viability of a system in the requirements and high level design stages. "Early feedback is crucial to building safe software" [7].

The importance of early feedback has been demonstrated empirically from both an economic and a safety point of view. Boehm showed that errors are cheaper to fix the earlier they are detected in the development lifecycle [8], while Lutz showed that requirements errors are more likely to be safety critical [9]. It is also clear that conventional techniques fail to catch many requirements errors [10]. However, it has not yet been demonstrated that formal methods offer a cost effective route to providing this feedback: the majority of requirements modeling studies have been post hoc reconstructions, in which results were not produced in time to affect the requirements definition phase of a project.

This paper describes three pilot studies in the application of formal methods to the Verification and Validation (V&V) of fault protection software on NASA spacecraft. Two of the studies concern the International Space Station, while the third concerns the Cassini deep space mission. For each study, we describe what methods were applied, how they were applied, how much effort was involved, and what the outcome was.

The three studies share a number of significant features:

1) Formal methods were applied in response to an existing development problem involving requirements. In each case the problem was to provide an assurance that the fault protection requirements were correct. Existing techniques could not provide this assurance.
2) Formal methods were applied selectively, in that only portions of the requirements of greatest concern were modeled, and only a selection of properties of these requirements were analyzed. The formal methods were applied by a research team working in parallel with the requirements analysts, rather than by the requirements analysts themselves.
3) In each case, formal methods offered a partial solution to the original problem. In particular, they provided a consistent requirements model, and revealed a number of errors, some of which could not have been detected using conventional approaches. However, in each case the formalisation was incomplete. The studies increased the confidence in the requirements, but did not guarantee the completeness and correctness of the specifications. We argue that this is appropriate for early modeling of requirements.
4) In each case, the results of the study fed back into development process to improve the product.

We summarize observations on the utility of formal methods in these studies, and describe problems we encountered in applying them. Finally, we describe our current work exploring application of formal methods in evolutionary design of new architectures for autonomous spacecraft control systems, and the special challenges of formally modeling evolutionary designs.

II. Background

1 Fault Protection

For NASA spacecraft, the term fault protection is used to describe system elements that avoid, detect and respond to perceived spacecraft faults. There are generally two over-riding requirements when a fault occurs: the system needs to guarantee the completion of any time critical activities, and it needs to guarantee that the spacecraft is still safe, observable and commandable. Various system level analyses are used to determine the possible faults that can occur, and some faults may be considered out of scope for a fault protection system, if they are regarded as too unlikely.

Fault protection software must be capable of monitoring the health of both hardware and software components, and detecting "out-of-tolerance" conditions, which may indicate the presence of a fault. In practice this is achieved by defining a set of operating parameters for each spacecraft function, where each parameter has a normal operating range. Values outside this range are *out of tolerance*. An out-of-tolerance condition may have many possible causes, so it is important to combine information from multiple sources in order to locate the fault. The values needed to determine out of tolerance values for each parameter are derived from the results of various system level analyses, including failure modes and effects analysis (FMEA), hazard analysis, and safety analysis. These analyses also provide rules of inference for fault recovery.

Fault protection software monitors for out of tolerance conditions, and initiates appropriate responses when such conditions are detected. Responses to loss of function include recovery (e. g. switch to a redundant backup), or retry (e.g. re-start a device in an attempt to restore functionality where no backup is available). Hazardous conditions generally require a 'safing' response, to isolate the problem and minimize damage. For unmanned spacecraft, a typical safing response is to shut down all non-critical functions, and ensure the antenna is pointing towards Earth, to await further commands from the ground. On Cassini, there is a requirement to be able to maintain such a safe state for up to two weeks. For manned spacecraft, such as the space station, there is a possibility of crew intervention, and a so a further requirement is to isolate the fault to the smallest possible replaceable unit.

Because of the need to maintain a safe, habitable environment for the crew, fault protection on the space station has additional requirements over those for unmanned craft, and the term Fault Detection, Isolation and Recovery (FDIR) is used in preference to 'fault protection'. Responsibility for FDIR is divided up into layers, or domains. The lowest domain is the individual device level. The next level is the function that uses the device. After that come the subsystem, and system control levels. The highest level is manual FDIR. If any particular domain cannot provide FDIR for some conditions, it must be provided by a higher level domain. For example, if an error condition involves the interaction of two separate devices, then FDIR might be provided by the subsystem level, rather than a the device level.

Fault protection software is a critical component of any spacecraft. As this software only executes when a failure has already occurred, it is important that the fault protection software responds correctly to the failure condition. If the spacecraft is executing a critical function (e.g. an orbital maneuver) when the failure occurs, it is also important that the fault protection responds quickly to allow the critical function to proceed.

Fault protection operates asynchronously, and may be invoked at any time. Hence, the addition of fault protection software to a spacecraft system significantly increases the behavioural complexity of the software. The wide range of possible interactions between fault protection and other systems makes it hard to verify the fault protection system. Furthermore, errors are more likely to occur during critical functions, because of the extra load on the spacecraft, and so fault protection software is more likely to be executed at the busiest times. An error in the fault protection software itself may compound an existing failure. This occurred on the initial launch of Ariane 5, when the fault protection software shut down two healthy computers, in response to an unhandled floating point overflow exception in a non-critical software function [11].

2 The Need for Formal Methods

Formal methods offer an excellent opportunity to advance the state of the art in V&V of requirements in areas such as spacecraft fault protection. Current requirements engineering processes within NASA rely extensively on manual procedures, largely based on inspection. Rigorous inspection processes help to remove a large number of specification errors, but cannot provide the desired level of assurance for mission critical software. Remaining errors are detected in an ad hoc way throughout the lifecycle as the developers attempt to implement and test the required system.

Requirements engineering processes within NASA appear to have reached a "quality ceiling" in which the currently employed development and assurance techniques have been optimized so much that no further improvements can be expected. This effect is shown in the data from formal inspections, in which the number of defects found in the requirements phase is seven times higher than in the code phase [10]. There is a significant lack of effective methods and tool support for the requirements phase in comparison to those available for detailed design and coding.

The lack of rigorous requirements engineering techniques is well illustrated in the fault protection area. Fault Protection requirements are derived from failure models of the target system, along with various safety analyses. From these sources, individual requirements are written to identify different fault conditions, initiate the appropriate response, and monitor the outcome. The results are expressed in a combination of tables, diagrams and prose, with an emphasis on prose for baseline requirements. The result is a large complex set of requirements documents, in which interactions between requirements can be hard to identify, let alone validate. Further problems arise from the fact that fault protection requirements are more volatile that most other requirements, as they are sensitive to any change during the development of the target system.

The complexity of fault protection means that it is hard to demonstrate that the system and software requirements for fault protection adequately describe everything that is needed to achieve the goal of providing robust spacecraft. Formal methods can help provide this validation in a number of ways. The process of formalising a specification provides a simple validation check, in that it forces a level of precision and explicitness far beyond that needed for informal representations. Once a formal specification is available, it can be formal challenged [3], by defining properties that should hold, and proving that they do indeed hold. Formal challenges may be achieved both through the use of mathematical proofs, and through state exploration or 'model checking'.

Rushby [3] points out that there is considerable scope for selective application of formal methods. For example, formal methods can be applied just to selected components of a system, and can be used just to check selected properties of that system. Most importantly, a great deal of benefit can be derived from formal methods without committing a project to the use of formal notations for baseline specifications. In the studies described in this paper, we used formal modeling to find errors in critical parts of existing informal specifications, but did not replace the informal specifications with their formal counterparts. This approach is consistent with the advocacy of multiple representations as a way of overcoming analysis bias.

3 Formal Methods and NASA

A multi-center team within NASA has been exploring the potential of formal methods [12, 13]. The team combines personnel with experience in formal methods, in the domains where formal methods are being applied, in software assurance and V&V, and in technology transfer. A series of studies by this team have explored formal methods on a number of NASA programs, including Space Shuttle [5], Space Station [14,15], and Cassini [16]. Throughout these studies, the emphasis has been on pragmatic application of formal methods in areas where there appears to be the greatest need. Results of these studies are described in two NASA guidebooks [17,18].

Although some development of the methods themselves has been necessary in order to fit them to our purpose, this has not been the main focus of the studies. Rather, we have concentrated on addressing issues such as:

- Can formal methods provide a cost effective addition to the existing techniques for improving the quality of requirements specifications?
- Can formal methods increase the confidence in the validity of the requirements?
- Can early application of formal methods be beneficial even while requirements are volatile?
- How much effort is needed to apply formal methods, and what is the most appropriate process for applying them?
- Within any particular formal methods process, which activities require more effort, and which activities yield the greatest benefits?
- Which formal methods and tools are useful for which tasks?

In this paper we describe the studies that were implemented in the early stages of requirements for new systems. To a large extent, these studies were responses to real needs on the projects. In each case the study was conducted in parallel with the requirements engineering process, so that results from the study could be fed back into that process. This meant that the requirements were often still volatile, and hence some effort was needed to ensure the formal analysis was kept up to date. However, we felt it was important to demonstrate that formal methods could be applied in this context, if we are ever to encourage wider adoption across the agency.

Although the three studies described here used different tools and notations, the basic approach is the same:

1) restate the requirements in a clear, precise and unambiguous format;
?) identify & correct internal inconsistencies
3) test the requirements by proving statements about expected behavior.
4) feed the results back to the requirements authors.

In two out of the three studies, step 1 involved an intermediate, informal notation, as a prelude to translating the requirements into the formal specification language. The intermediate notation helped to clarify ambiguities, and gain a better understanding of the structure 01 the requirements. This in turn helped to determine how the formal notation would be 11 sed.

Study 1: High level FDIR requirements for Space Station

The purpose of this study was to assist with the independent assessment of the fault detection, isolation and recovery (FDIR) requirements for the space station. Verification of the space station 1 DIR systems is particularly problematic, as FDIR functionality is distributed across many of the fli.gilt computers. The development and construction schedule for the space station dots not permit full integration testing of the entire architecture prior to on-orbit assembly. Hence, FDIR functionality must be verified through a combination of inspection, testing and analysis.

Independent assessment is an oversight activity, covet ing all aspects of the system, including hardware, software and operational procedures. The aim is to assure an appropriate level of safety in the development of the space station. At the time of this study, the independent assessment panel was seeking some assurance that the high level FDIR concept was clearly defined and validated, before it flowed down to end item requirements. Subsequent changes to the FDIR concept would have significant impacts throughout the requirements and design of the entire system. For these reasons, the independent assessment panel commissioned a formal analysis of the high level FDIR function. The study was jointly funded by NASA headquarters, as part of the pilot program in formal methods.

The need that arose from the independent assessment dovetailed with the aims of the inter-center formal methods team. We had completed some preliminary studies of Space Shuttle re-engineering requirements [5], which had demonstrated the potential for formal methods as a requirements assurance technique. However, this work concentrated on analyzing change requests for an existing system. The space station work was a chance to get in early in the high level (system) requirements phase for an entirely new system. We needed to investigate whether there were any significantly different problems associated with applying formal methods to the early modeling activities in a requirements phase for a new system.

1 *Approach*

Three views of' the FDIR had been documented: the functional concept diagram (FCD) which is a flowchart like representation of the generic FDIR algorithm; baseline FDIR requirements; and capabilities, in which the requirements are grouped into related functional areas. This study concentrated on the first two of these views, developing a formal model of each, and testing traceability between them.

The four step approach described above was used. In this study, restating the FCD involved a process of abstracting out common features before it could be translated into PVS [19]. The baseline requirements were translated directly into }'VS. I'VS was chosen for this study, because it provided an automated theorem proving support, and because the specification language appeared to be readily understandable to engineers and programmers. Internal consistency of the models was tested using PVS typechecking, while the expected behavior was analyzed by defining theorems expressing required properties, and showing that they followed from the model using the PVS proof assistant.

The first step was to analyze the FCD. The original FCD contained 53 processing steps, making it rather complex. As a first step in the analysis, this diagram was partitioned, in order to create a more abstract view. For example, the first 12 steps involved checking parameters for out of tolerance conditions, the next 7 dealt with safing, the next 8 dealt with checking for functional failure, and so on. In addition, each step was labeled as one of three procedural categories: performing automated procedures, checking for anomalous conditions, and recording/reporting results. Finally, the conditions under which control is passed to higher level FDIR domains were identified. Six categories of condition under which this occurs were identified. The result of this initial analysis was a more structured (informal) model of the FDIR processes. This model was informally checked for reasonableness, and for traceability to the original FCD. A number of anomalies were discovered at this stage, which were reported to the requirements authors

The next step was to formalise the model in PVS. A consistent terminology was developed, and all objects and attributes referenced in the FCD were expressed in PVS. Figure 1 shows two fragments of PVS generated at this stage. The resulting definitions were typechecked using the PVS tool. Typechecking helped to eliminate several types of errors in the specification, including typos, syntax errors and type consistency errors.

```
message: type =
{  parameter_OK,
   parameter_verified,
   safing_not_allowed,
   safing_executed,

)

% parameter is ok when its tolerance
% check has just ran and the parameter
% is OK (i.e. within tolerance)
rr_parameter_ok: axiom
   forall (t: tolerance_check):
   (  on(just_ran(t, time) and
      OK?(t(time)))
   iff
      record_check(time)(parameter_OK, t)
   )
```

Figure 1: Fragments of PVS specification, showing type definitions and axioms used to express FDIR concepts

Finally, the PVS specification was validated by using the PVS proof assistant to prove claims based on the specification. An example of such a claim is "at any domain level, if a failure occurs then it will always be recovered at some domain level". Although this claim was not very profound, several missing assumptions were detected in the process of proving it. For example, several sequencing constraints needed to be defined explicitly, even though the FDIR documentation had stated that no such constraints should be inferred from the requirements. A total of 14 claims were defined and proved. Most of these were type correctness conditions (TCCs), which mainly serve to ensure internal consistency of the model.

The second part of the study was to analyze the baseline system requirements for FDIR. A distinction can be drawn between the primary space station system, and the FDIR system that monitors the primary system. The formal modeling concentrated only on the latter. The prose requirements were translated into PVS, using the definitions and types generated in the first part of the study. Translation of these requirements into PVS proved to be relatively straightforward. Figure 2 gives an example.

```
Requirement: automatic hazard and hazardous condition detection: ISSA
shall automatically detect any out-of-tolerance condition or functional
performance parameter that exhibits a time to catastrophic or critical
effect of less than 24 hours.

automatic_hazard_condition_detection: axiom
   forall (p:parameter)
      param_out_of_tol?(p) AND time_to_effect(p)<24 =>
      exists(d:fdir_domain): detection(p,d) = automatic
```

Figure 2: An example FDIR requirement, and its PVS translation

The process of translating these requirements revealed a number of relatively minor ambiguities and incompletenesses. For example, the distinction between the primary system and the FDIR system was not clear in the original requirements. Other ambiguities surrounded the use of terms such as "anomaly", 'out-of-tolerance" and "functional failure".

Having modeled both the FDIR concept diagram and the baseline requirements, the plan was to explore traceability between the two. An initial analysis indicated that there was little traceability. The requirements authors confirmed that the Iwo documents expressed different kinds of requirements. The FCD describes the processing that is performed within an FDIR domain, while. the baseline requirements describe a higher level view of the kinds of FDIR that must be provided.

2 *Findings*

In general, the FDIR requirements were well thought out. However, although the FDIR requirements team understood them, there \vas some question over whether the documentation was sufficient so that system developers and other stakeholders would understand them. A total of fifteen issues were documented and discussed with the requirements authors. Most of these were minor ambiguities, inconsistent use of terms, and missing assumptions, discovered during the process of formalisation, which reduce the ability 01 developers to understand the requirements. Three of the issues were regarded a\ "high-major":

I) There were inconsistencies in the FCD over reporting the status of safing, recovery and retry procedures. The intention was that the FDIR processes should report their status before, during and after execution of each procedure. However, some of the procedures were missing requirements for some of the reporting activities, so that most of them did not have requirements to report status at all three points. This was detected during the initial analysis of the FCD diagram.

?) The proper sequencing of FDIR processing is not clear from FCD. Although the FCD looks like a flowchart, the accompanying text makes it cleat it should not be interpreted as a sequential process. However, some important requirements can only be inferred by treating it as a sequential process. For example, it is not **clear** whether safing should be performed before isolation, although the diagram seems to imply it should be. 'Ibis problem was detected during the proof process: some of the sequencing requirements had to be slated explicitly in order to prove necessary properlies of" the FDIR model.

3) No requirements are given for checking inconsistencies between parameters; the requirements only mention limit checking of individual parameters. The requirements team clearly intended that inconsistency checking should be included. This problem was discovered during the process of formalising the baseline requirements.

3 *Observations*

The study analyzed 18 pages of FDIR requirements, and was conducted over a period of two months, by two people working part-time. The total effort was approximately ? person-months. Reformulation of the FCD was the most cost effective part of the study. Typechecking of the PVS specifications was also relatively inexpensive, and useful primarily to remove mistakes that weir introduced in the translation process. The proof exercise was costly (mainly because of the expertise needed), but paid off in terms of checked assumptions, and confidence in the accuracy of the model. The effort and tirnc.scale of the study were consistent with the normal V&V processes for the requirements phase. In this study, formal analysis provided a cost-effective enhancement to existing practices.

The requirements' authors reviewed the results of the study. Some of them also reviewed the approach and the resulting formalisation in detail. They had a strong desire to make sure the requirements were clear and unambiguous Many of the findings of the study coincided with the types of questions that were beginning to arise from the teams charged with implementing FDIR. '1'0 some extent the requirements ' authors wanted more than this study could offer: they wanted to know whether the FDIR as specified would work correctly. A follow-up study is being conducted to help address this question, but is not complete at the time of writing.

Study 2: Detailed Bus FDIR requirements for Space Station

This study can be seen as a natural follow on to the previous one, although it was not originally planned as such. The purpose of this study was to analyze the detailed FDIR requirements associated with the bus controller for the main 1553 communications bus on the space station. These requirements represent a concrete implementation of the high level FDIR coin-JIls addressed in the first study.

The study was initiated by the Independent Verification and Validation (IV&V) team. IV&V is a practice in which a separate contractor is hired to analyze the products and process of the software development contractor **[20]**. The IV&V team was having particular difficulty validating the bus FDIR requirements, as they were hard to read, and some of the properties they wished to test could not be established using existing in formal methods. The study was conducted by the research team, as part of a larger study of the use of multiple representations in the V&V process.

The requirements for Bus FDIR were expressed in natural language, with a supporting flowchart showing the processing steps involved. The flowchart did not have the status of a requirement, but was merely provided for guidance; the intention was that the prose completely expressed the requirements. The prose contained a number of long complicated sentences, expressing complex conjunctions and disjunctions of conditions. The IV&V team had recommended that to improve clarity, the requirements should be re-written in a tabular form (specifically, as truth tables similar to those used in [21]). This recommendation had been rejected because of the cost involved in re-writing them all. Hence, the IV&V team generated their own tabular versions, in order to facilitate the kinds of analysis they wished to perform.

1 Approach

The four step approach was used as follows. Each individual requirement was restated as a truth table, to clarify the logic. These were then combined into a single state-machine model, using SCR [22]. SCR was chosen for this study as it offered a tabular notation that corresponded well to the truth tables that the IV&V team had already adopted, and it provided tool support for checking consistency of SCR models. Consistency checking involved type checking of the SCR specification. Properties of the model were then tested in two ways. First, static properties of the state model, such as disjointness and coverage, were tested using the built-in checker in the SCR tool. Second, dynamic properties of the model were tested by translating the SCR state. machine model into PROMELA [23], and applying the SPIN model checker to explore its behavior.

The generation of a tabular interpretation of each individual requirement proved to be hard, as there are a number of ambiguities in the prose requirements. These ambiguities concern the associatively of 'and' and 'or' in English, and the correct binding of subclauses of long sentences. For example, in figure 3, it is not clear what the phrase "in two consecutive processing frames" refers to. To confirm the existence of such ambiguities, the requirement shown in Figure 3 was given to four different people, for translation into tabular form. Four semantically different tables resulted. By comparing these different interpretations, an extensive list of ambiguities was compiled. The ambiguities were resolved through detailed reading of the documentation, and questioning the original authors. This process also revealed some inconsistencies in the way in which terminology was used.

```
(2.16.3.f) While acting as the bus controller, the C&C MDM CSCI shall
set the e,c,w, indicator identified in Table 3.2.16-11 for the
corresponding RT to "failed" and set the failure status to "failed" for
all RT's on the bus upon detection of transaction errors of selected
messages to RTs whose 1553 FDIR is not inhibited in two consecutive
processing frames within 100 millisec of detection of the second
transaction error if; a backup BC is available, the BC has been switched
in the last 20 sec, the SPD card reset capability is inhibited, or the
SPD card has been reset in the last 10 major (10-second) frames, and
either:
1. the transaction errors are from multiple RT's, the current channel
has been reset within the last major frame, or
2. the transaction errors are from multiple RT's, the bus channel's
reset capability is inhibited, and the current channel has not been
reset within the last major frame.
```

Figure 3: An example of a level 3 requirement for Bus FDIR. This requirement specifies the circumstances under which all remote terminals (RTs) on the bus should be switched to their backups.

		OR			
	C&C MDM acting as the bus controller	T	T	T	T
	Detection of transaction errors in two consecutive processing frames	T	T	T	T
	CRCs are on selected messages	T	T	T	T
	the RT's 1553 FDIR is not inhibited	T	T	T	T
	A backup BC is available	T	T	T	T
A	The BC has been switched in the last 20 seconds	T	T	T	T
N	The SPD card reset capability is inhibited	T	T	•	•
D	The SPD card has been reset in the last 10 major (10 second) frames	•	•	"/"	T"
	The transaction errors are from multiple RTs	T	T	T	"1"
	The current channel has been reset within the last major frame	T	F	T	F
	The bus channel's reset capability is inhibited	•	T	•	T

Table 1: The tabular version of the requirement shown in figure 3, showing the four conditions (the four columns) under which the action should be carried out. A dot indicates "don't care".

Having obtained a clearer statement of the requirements, the next step was to explore some of the properties that ought to be true of these requirements. Example properties are "for each combination of failure conditions, there is an FDIR response specified" and "for each combination of failure conditions there is at most one FDIR response specified". These properties correspond to checks for coverage and completeness of a mode table in SCR. Hence, by constructing a state based model in which each of the requirements represented a transition from the "normal" mode to a unique failure mode, the coverage and disjointness tests provided in the SCR tool would test these properties. As a result, a number of disjointness problems were identified, which are described below.

The final part of this study was to explore some of the dynamic properties of the model. For example, some of the requirements express conditions that test whether various recovery actions have already been tried. In order to validate these conditions, it was necessary to explore the dynamic behavior of the specified system in the face of multiple failures, and recurring failures. To do this, the state-based model expressed in SCR was translated into PROMELA, and the model checker SPIN was used to explore the behaviors. The translation into PROMELA indicated some inconsistencies in the timing constraints that had not been revealed in the SCR model. Once these were fixed, the model was checked for properties such as "if an error persists after all recovery actions have been tried, the bus FDIR will eventually report failure of itself to a higher level FDIR domain".

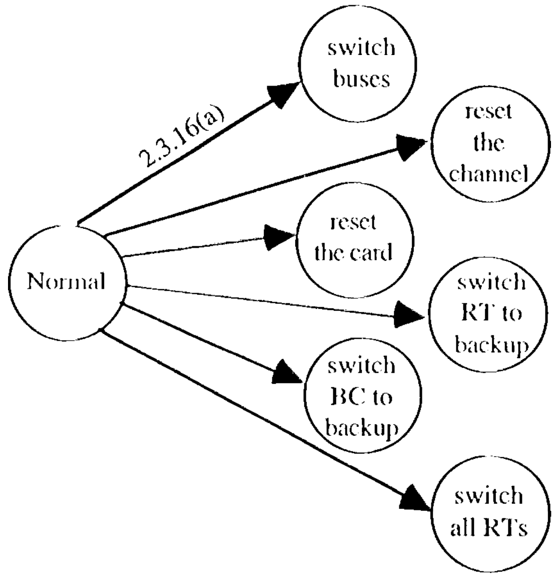

Figure 4: A graphical representation of the SCR mode table. Each requirement specifies the conditions for a single mode transition.

2 *Findings*

In addition to a number of minor problems with inconsistent use of terminology, the following major problems were reported:

1) There were significant ambiguities in the. prose requirements, as a result of the complex sentence structure. Some of these ambiguities could be resolved hy studying the higher level FDIR requirements, and the specifications for the bus architecture. The ambiguities that arose from the combination of 'ands' and 'ors' in the same sentence could not be resolved in this way, and could lead to mistakes in the design. These ambiguities were detected in the initial reformulation of the requirements as truth table.s.

2) There was one missing requirement to test the value of the Bus Switch Inhibit Flag before attempting to switch to the backup bus. This was detected during the test for disjointness in the SCR specification.

3) The prose requirements were missing a number of preconditions that enforce the ordering in which the inference rules should be applied. The accompanying flowchart for these requirements implied a sequence. for these rules. An attempt had been made in the prose requirements to express this sequence as a set of preconditions for each rule, to ensure that all the earlier rules have been tested and have failed. The preconditions did not completely capture the precedences implied by flowchart. This corresponded with an informal observation made hy the IV&V team that the ordering of the requirements should be made explicit. This problem was found during the test for disjointness in the SCR specification.

4) The timing constraints expressed in the requirements were incorrect. Several of the failure isolation tests referred to testing whether certain FDIR actions had already been tried "in the previous processing frame". However, as each FDIR recovery action is followed by a time-out in order for the action to take effect, and as further FDIR intervention is only initiated on occurrence of errors in two consecutive processing frames, these tests earl never he true. This was discovered during model checking of the PROMELA model.

3 *Observations*

The study analyzed 15 pages of level 3 requirements, and was conducted over a period of four months, by one person working part time. The total effort was approximately 1.5 person months. The main effort was in formalising the requirements. Translation from the SCR model to PROMELA was relatively straightforward, and took two days effort. Once a formal model was obtained, testing of the properties was straightforward, as both the SCR tool and the SPIN model checker provided facilities for automated checking of these properties, and provided counter-examples when the tests failed. Although problems were found both during formalisation and the property checking, the latter problems were more serious. II is unlikely that they would have been discovered in this phase without the usc of formal methods.

A major problem during this study was the volatility of the requirements. Nc-w drafts of the requirements document were being released approximately every two months. This meant that in at least onc case (finding 3 above), the. problem had already been fixed try the lime it was discovered in this study. This issue had already been observed informally and reported by the IV&V team, and had been addressed hy reducing the complexity of this section of the requirements. We mitigated the problem of fluctuating requirements hy only doing the minimum amount of modeling necessary to test the properties that **were** of interest. For example, **the** SCR model is not a complete state model, as it models only a subset of the slate transitions expressed in the requirements. The transitions for returning to the normal **state** have not **been** modeled. This partial model was sufficient to perform the coverage and disjointness analysis.

It should also be noted that in order to perform the analysis in this study, the SCR notation" was slightly misused. The modes shown in figure 4 (k) not represent true modes in the SCR sense - a more correct representation would express these as output events from the FDIR system. However, defining them as modes permitted the usc. of coverage and disjointness tests on the. transitions. This represents a pragmatic approach in which the formal method is applied in whatever way gives the most benefit, without necessarily following the original intent of the method.

Study 3: Fault Protection on Cassini

The third study concerns the system level fault protection software for Cassini. Cassini is a deep space probe, to be launched in 1 997, which will explore Saturn and its moons. System reliability is a major concern for Cassini, due to the duration of the mission. Fault protection is a major factor in providing the required levels of reliability. Fault protect ion soft ware is therefore mission-crit **ical**, in addit ion to being a Complex embedded system. The st udy examined the. requirements for two main components of the fault protection system: the software executive that manages fault protection, and requirements for putting the spacecraft into a safe state.

The aim of this study was to explore the effectiveness of formal methods in supplementing traditional engineering approaches to requirements analysis. The Cassini project was interested in the potential of formal methods to provide an assurance that the fault protection requirements were correct, while the formal methods team was interested in the opportunity to apply formal methods early in the requirements process, where tally modeling of unstable requirements might pose a challenge.

1 Approach

For this study, the initial step of re-stating the requirements included the use of OMT diagrams. These were. them used to guide the development of a PVS model of the requirements. Once the PVS model was checked for internal consistency, a number of properties were defined, to check that the software would function correctly and be hazard free.

The first step was the production of OMT diagrams representing the documented requirements (see figure 5). The original requirements were expressed in natural language. The production of object diagrams, state diagrams and dataflow diagrams, according to the OMT method, helped to define the boundaries and interfaces of the fault protection requirements, and helped to crystallize some of the issue.s that arose in the initial close reading of the. requirements. A number of issues having to do with imprecise terminology, inconsistency between text and tables, and unstated assumptions were discovered during the OMT modeling.

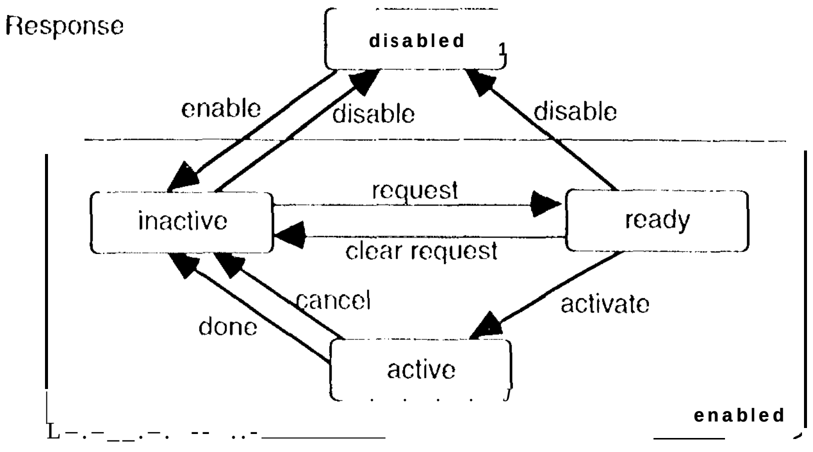

Figure *5:* **An example OMT state diagram for fault protection**

The OMT diagrams them served as a frame for the subsequent formalisation. **A** PVS model was produced directly from the OMT models - the elements of the OMT model often mapped onto elements of the formal model in a relatively straightforward way. For example, object classes mapped onto type definitions in PVS, while state transitions mapped onto functions and axioms.

Once the I'VS mode] was completed, a number of lemmas were defined to examine various properties of the requirements. Three different categories of property were examined:

1) requirements-met. These lemmas helped to trace the model back 10 the documented requirements, and ensure that the model accurately captured the documented requirements (see figure 6). For example, a requirement "If a response can be initialed by more than one. monitor, each monitor shall include an enable/disable mechanism" was expressed as a lemma, to test whether the model met this requirement. in this category, seven lemmas were proved, and three. disproved.

2) safely. These lemmas represented conditions that should not arise, (o lest that "nothing bad ever happens". For example, "A fault protection response shall not change the instrument's status during a Critics] sequence of commands". Seven of these lemmas were proved.

3) liveness. These lemmas describe the correct behavior, i.e. that "something good will eventually happen". An example is "If a response has the highest priority among the candidates and dots not finish in the current cycle, it will be active in the. next cycle". Seven of these lemma\ were proved.

```
Cassini Requirement: If Spacecraft Safing is req uested v i a a CDS
(Command and Data Subsyst em ) int ernal request whi le the spacecraf t is in
a critical at. t i tude, then no change is commanded t o the AACS (At tit ude
and Articul ation Cont rol Subsyst em) at t i tude. Ot herwi se, the AACS i s
commanded t o the hom ebase at t i t ude .

saf: THEORY
% Example is excerpted from saf theory.
% Spacecraft safing commands the AACS to homebase mode, thereby
% stopping delta-v's and desat's.
BEGIN

aacs_mode:   TYPE = {homebase, detumble}
attitude:    TYPE

cds_internal_request :     VAR bool
critical_attitude:         VAR bool
prev__aacs_mode:           VAR aacs_ mode

aacs_stop_fnc (critical. attitude, eels. internal_request , prev_ sac-s. mode):
    aacs_ mode =
    IF critical_ attitude
        THEN IF eels. internal. request
            THEN prev_ sacs. mode
            ELSE home base
            ENDIF
        ELSE homebase
    ENDIF

% Lemma prove n, providing assurance that PVS speci f i cat i on mat. ches
% documented re qui rement .

aacs_ safing_ req met_ 1 :   LEMMA
    (critical_attitude AND cds_ internal. request-)
    OR (sacs. stop_ fnc (critical _ attitude, eels. internal_ request ,
        prev_ aacs_mode ) = homebase )

END saf
```

Figure 6: An example Cassini fault protection requirement, a fragment of PVS representing this requirement, and an associated 'requirements-met' lemma.

2 *Findings*

A total of'37 issues were identified during the study. These were classified as follows:

Undocumented assumptions: 1 1. All were conect, but some significant ones needed documentation, to prevent future errors, especially at intci-faces, These were identified during the process of formalising the requirements.

Inadequate requirements for off-nominal or boundary cases: 1 (). These issues usually involved unlikely scenar ios, and the spacecraft engineers had to help decide which were credible. An example case is when several monitors with the same priority level detect faults in the same cycle. Documentat ion of these cases is useful, as it helps to verify the robustness of the system.

Traceability and inconsistenc y: 9. There we r e a number tr actability problems between different levels of requirements, and inconsistencies between requirements and subsystem desig ns. Many of the latter were significant, as the correct functioning of the system depends on choosing" the comet interpretation. For example, the high level requir ements ass ume that multiple detections of faults, occurr ing within the response time of the first fault are symptoms of the original fault, whereas the lower level requirements correctly cancel a lower priority fault response to handle a higher-priority one.

Imprecise terminology: 6. These were largely documentation problem, including synonyms and related terms, and were revealed during the process of defining the PVS *model.*

Logical Error: I. This was a problem of starvation when a request for service is pre-empted by a higher priority request. The issue was first spotted during initial close reading, and confirmed by disproving a lemma.

3 *Observations*

The study analyzed eighty-five pages Of documented requirements. Fifteen pages of OMT diagrams were produced, followed by (wenty-five pages of J'VS specifications. Twenty-four lemmas were proven. The study was conducted over the period of a year by two people working part-time, with a total effort of approximately twelve person-months. The main effort came in learning to effectively usc the PVS theorem prove].

OMT models were found to be useful as a first step in developing formal specifications. The OMT diagrams bounded the software at an appropriate level of abstraction, offered multiple perspectives on the requirements, and were easy for Project personnel to review for accuracy. Since the elements of OMT diagrams often mapped directly onto elements of the formal specifications, the subsequent effort of formalization was reduced. Iterations of the OMT and PVS models still occurred while proving claims about the model, but the conciseness and readability of the. OMT notations made it easier to confirm the accuracy of the models. In effect the. OMT model provided a higher level structural view of the requirements, while the PVS models filled in the. processing details, and allowed detailed behavioral analysis.

The requirements that were analyze.d were known to be influx with several key areas (e.g. timing, number of priority levels) still being determined. Time was spent keeping the models current with the updated requirements. This extra effort was balanced against the advantage that issues identified by the formal analysis were readily fed hack into the development process, leading to improved requirements.

A key concern of the rc.searchers was whether formally modeling requirements that were known to be unstable was a waste of effort. In general, **the** effort in this study was found to be worthwhile because the modeling so effectively laid the foundation for future work, allowing rapid response to proposed changes or alternatives by the Cassini Project. in addition, the work bad the anticipated advantage of adding confidence in the adequacy of the requirements that had been analyzed using formal methods. in some cases where requirements issues were. still being worked by the Project, the formal methods effort was able to assist by formalizing undocumented concerns (e.g., whether starvation of tasks would be possible) clearly and unambiguously.

V. Discussion

The studies described in this paper differ from previous studies in the literature in several ways. The majority of published case studies of the usc of formal methods are post hoc applications to on-going or finished projects. Such studies demonstrate what formal methods *can* do, and help to refine the methods, but they do not help to answer questions of how such methods can be integrated with existing practices on large projects. A few notable exceptions have used forma] methods 'live' during the development of real systems [], 2 1, 24, 25]. However, in all the.sc cases, the emphasis was on the use of formal notations as a part of the baseline specifications, from which varying degrees of formal verification of the resulting design and implementation are possible.

In contrast, we applied formal methods only in the early stages of requirements engineering, during which the requirements were still volatile. Rather than treating formal specification as an end product of the requirements phase, we used it to answer questions and improve the quality of the existing specifications.

In the longer term we hope to introduce formal methods throughout the lifecycle. Our experiences with these studies indicate (hat we can best do this incrementally, in response to real needs in specific projects.

Our approach dots not fit with any of the three process mode.ls suggested by Kemmerer [25] as ways of applying formal methods. Kemmerer offers three alternatives: nj'let-fhe-~(wf, in which a formal specification is produced at the end of the development process to assist with testing and certification; *parallel*, in which formal specifications are developed alongside a conventional development process, and used to perform verification of code, design and requirements; and *integrated*, in which formal specification is used in place of conventional approaches, Our studies suggest a fourth model, in which formal modeling is used lo increase quality during the requirements and high level design phases, without necessarily producing a baseline formal specification, or verifying low level design and code

Our studies also demonstrate that questions of tool support need not be a barrier to the adoption of formal methods. We conducted sophisticated validation of our models, via theorem proving and model checking, using tools that arc essentially still research prototypes. In the 12 case studies surveyed by Gerhart *et. al.* [24], tool support was generally only used for syntax checking of specifications, and Gerhart suggests tool impoverishment is a barrier to wider use of formal methods. This may be true for the more complete process models used in case studies of the kinds described by Kemmerer [25], Hall [1] and Gerhart [24], but is not true of the 'lightweight' application of the kind wc adopted.

Most of our observations of the benefits of formal methods are consistent with findings elsewhere. For example, wc noted that a large number of minor problems are discovered during the process of formalising the requirements, and that the use. of formal methods helps to focus attention on areas that are more susceptible to errors [26], Formally challenging the models uncovered a smaller number of more subtle issues, of the kind that arc hard to detect manually. Like Hall [1], wc found that the usc of intermediate , structured representations greatly facilitated the process of formalising tbc requirements.

Although wc have not attempted any detailed quantitative analysis of the costs and benefits of the application of formal methods in these studies, it is clear that in each Case the study added value to the project by clarifying tbc requirements and identifying important errors very early in the lifecycle. The costs, in terms of time and effort, were consistent with existing V&V tasks on these projects.

A number of observations arising from these studies at-e worth further discussion:

Who should apply the methods?

in each of the studies, the formal analysis was conducted by experts in formal methods, who were external to tbc development project. There was a simple financial reason for this: it is cheaper and lower risk to have a small team of formal methods experts develop the specifications and perform tbc analysis than it is to train members of the development team. Out longer term goat is to have the developers produce formal specifications themselves, with a V&V team performing the analysis.

} lowever, there arc some interesting consequences of our use of external experts 1)eveloping formal models of informal specifications involves a great deal of effort in understanding the domain, anti figuring out how to interpret the documentation. As our external experts were unfamiliar with the projects prior to the studies, they did not share the. assumptions that the requirements' authors bad made. Our experts questioned every thing, spurred on by the explicitness needed to build the formal models, They also needed to present parts of their models back to tbc developers, in order to check the accuracy of their interpretations. The result was a healthy dialogue between the developers and our formal methods experts. This dialogue exposed many minor problems, especially unstated assumptions and inconsistent usc of terminology. This dialogue was clearly an important benefit.

Another aspect of this dialogue was that some of the issues that were raised were the result of misunderstan dings by our experts, rather than genuine errors. The require.rncnts authors therefore had to filter the issues, to pick out those for which the benefits of changing the requirements out-weighed the cost. 'l'his was especially true when the analysis revealed "interesting" off-nominal cases. A great deal of domain knowl edge was needed to judge whether sue}) cases were r easonable. The need for such filtering would be greatly reduced if the analysis was conducted by domain experts; however, the risk of analysis bias would then increase.

Is formal modeling of volatile requirements worthwhile?

During early stages of the requirements process, there may be a great deal of volatility. In each case study, some effort was needed to keep tbc formal model up to date with evolving, requirements. However, the studies indicate that there is no need to wait for the requirements to stabilize before applying formal methods. Early forr nalisation allowed US to crystallize some of the outstanding issues, and explore different options. Most importantly, it is during this early phase that the development team is most receptive to the issues raised from tbc formal modeling. This again emphasizes the importance of lightweight formal methods: tbc formal model itself can be discarded if the requirements change significantly, while the experience and lessons learned from it arc retained.

• . "

Were intermediate representations useful?

Intermediate representations were an important part of the formalisation process in each study. The type of intermediate representation varied across the studies: the first study used an annotated version of the original FCD flowchart, the second study made use of truth tables to clarify complex predicates, while the final study made extensive use of OMT diagrams. A large part of the effort in the formalisation process lies in understanding the existing requirements. These intermediate representations helped to refine this understanding, and therefore reduced the effort needed to generate and debug the formal models.

The intermediate representations also helped to create some initial structure for the formal models. They assisted with traceability between the formal and informal specifications, making it simpler to keep the formal model current. From our experience thus far, it seems that this benefit more than outweighs the extra cost of maintaining several representations, even during the early stages when requirements are most unstable,

V1. Conclusions

The three studies described here were conducted as pilot studies to demonstrate the utility of formal methods and to help us understand how to promote their use across NASA. An important characteristic of these studies is that in each case the formal modeling was carried out by a small team of experts who were. not part of the development team. Results from the formal modeling were fed back into the requirements analysis phase, but no attempt was made to introduce formal specification languages for baseline specifications.

Wc have shown that lightweight formal methods complemented existing development and assurance practices in these projects. If formal methods is seen as an additional tool in the V&V toolbox, then widespread application to existing large projects becomes feasible.

As a follow-up to the studies described here, wc have begun to investigate the role of formal methods in the development of new spacecraft technology. As part of NASA's Ncw Millennium program, ncw architecture.s arc being developed using knowledge based systems to reduce the reliance of the spacecraft on ground support. Rather than produce a detailed statement of requirements, the project is using a rapid prototyping approach to explore the capabilities of the technology. The prototypes arc tested against high level objectives, using a set of high level scenarios for guidance.. We are exploring how to use lightweight formal analysis on rapidly changing information, in such a way as to provide useful and timely feedback. In particular, wc are exploring the use of mode] Checking 10 verify the fidelity between a formal model and the prototype. The model checker tests whether the formal model behaves in the same way as the prototype for a given scenario, while the. formal model can be used to find interesting ncw scenarios on which to exercise the prototype.

Bibliography

[1] A. Hall, "Using formal methods to develop an ATC Information System," *IEEE Software*, vol 13, pp. 6()-76, 1996.

[2] H. Saiedain, J. P. Bowen, R. W. Butler, D. L. Dill, R. L. Glass, A. Hall, M. G. Hinchey, C. M. Holloway, D. Jackson, C. B. Jones, M. J. Lutz, D. L. Parnas, J. Rushby, J. Wing, and P. Zave, "An invitation to Formal Methods," *IEEE Computer*, vol. 29, pp. 16-30, 1996.

[3] J. Rushby, "Formal Methods and q'heir Role. in the Certification of Critical Systems," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report CSL-95-1, March 1995.

[4] J. Crow, "l;inite-State Analysis of Space Shuttle Contingency Guidance Requirements," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report SRI-CSL-95-17, December 1995.

[5] J. Crow and 11. 1.. Di Vito, "Formalizing Space Shuttle Software Requirements," presented at Workshop on Formal Methods in Software Practice (FMSP '96), San Diego, California, January 1996.

[6] B. L. Di Vito, "Formalizing Ncw Navigation] Requirements for NASA's Space Shuttle," presented at Formal Methods Europe (FME '96), Oxford, England, March 1996.

[7] N. G. Leveson, *Safeware: System Safety and Computers*. Reading, MA: Addison Wesley, 1995.

[8] B. W. Boehm, *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[9] R. R. 1.1117, "Targeting Safety-lWlatcd Errors During Software Requirements Analysis," presented at Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering, 1993.

[10] J. C. Kelly, J. S. Sherif, and J. Hops, "An Analysis of Defect Densities Found During Software Inspections," *Journal of Systems and Software*, vol. 17, pp. 111-117, 1992.

[11] J. L. Lions, "ARIANE 5 Flight 501 Failure: Report by the Enquiry Board," European Space Agency, Paris 19 July 1996.

[1?] R. W. Butler, J. L. Caldwell, V. A. Carreno, C. M. Holloway, 1'. S. Miner, and 11. L. Di Vito, "NASA Langley's Research and Technology Transfer Program in Formal Methods," presented at Tenth Annual Conference on Computer Assurance (COMPASS 95), Gaithersburg, MD, June 1995.

[13] D. Hamilton, R. Covington, and J. C. Kelly, "Experiences in Applying Formal Methods to the Analysis of Software and System Requirements," presented at IEEE Workshop on Industrial-Strength Formal Specification Techniques (WIFT '95), Boca Raton, FL, April 1995.

[14] 1). Hamilton, R. Covington, and A. Lee, "An Experience Report on Requirements Reliability Engineering Using Formal Methods," presented at IEEE international Conference on Software Reliability Engineering, France, October I 995.

[15] S. Easterbrook and J. Callahan, "Formal Methods for V&V of partial specifications: An experience report," presented at Proceedings, Third IEEE Symposium cm Requirements Engineering (RE '97), Annapolis, Maryland, 5-8 January 1997.

[16] Y. Ampo and R. R. Lutz, "Evaluation of Software Safety Analysis using Formal Methods," presented at Foundation of Software Engineering '95, Hamana-ko, Japan, Dec. 14-16, 1995.

[17] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems. Vol 1: Planning and Technology Insertion," NASA Office of Safety and Mission Assurance, Report NASA-GB-002-95, 1995.

[18] NASA, "Formal Methods Specification and Verification Guidebook for Software and Computer Systems. Volume 2: A Practitioner's Companion (DRAFT)," NASA Office of Safety and Mission Assurance, Report NASA-G]%-???, 1996.

[19] S. Owre, J, Rushby, N. Shankar, and F. von Henke, "Formal Verification for Fault Tolerant Architectures: Prolegomena to the Design of PVS," *IEEE Transactions on Software Engineering*, vol. 21, pp. 107-1 ?5, 1995.

[20] S. Easterbrook and J. Callahan, "Independent Validation of Specifications: A coordination headache," presented at Proceedings, IEEE Fifth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'96), Stanford, CA, Jun 19-21 1996.

[21] M. Heimdahl and N. Leveson, "Completeness and Consistency Analysis of State-llascd Requirements," *IEEE Transactions on Software Engineering*, vol. 22, pp. 363-377, 1996.

[22] C. 1. Heitmeyer, B. Labaw, and D. Kiskis, "Consistency Checking of SCl{-Style Requirements Specifications," presented at Second IEEE Symposium on Requirements Engineering, York, UK, 1995.

[?3] G. J, Holtzmann, *Design and Validation of Computer Protocols*: Prentice Hall, 1991.

[24] D. Craigen, S. L. Gerhart, and '1'. Ralston, "Formal Methods Reality Check: industrial Usage," *IEEE Transactions on Software Engineering*, vol. ? 1, pp. 90-98, 1995.

[25] R. A. Kemmerer, "Integrating Formal Methods into the Development Process," *IEEE Software*, vol. 7, pp. 3"/-50, 1990.

[?6] 1'. G. Larsen, J. Fitzgerald, and "1'. Brookes, "Applying Formal Specification in industry," *IEEE Software*, vol. 13, pp. 48-56, 1996.